**Tokenisation**

The following tokens are stored in low/high order. Each is 16 bits wide except for that representing an ASCII string.

End Token : 0000 0000 0000 0000 $0000

This token is used to mark the end of a sequence of tokens (e.g. a line of tokenised code)

ASCII String : 0000 0000 LLLL LLLL $0000-$00FF

This represents a single ASCII string. The token identifies it as a string and gives the overall length in bytes of the string. This is not the same as the length of the string. It also includes the header word (2 bytes/1 word) and a leading length byte. The latter is mandatory. The string is stored in such a way that the address of the following token is itself the string prefixed with the length. If there is an even length string (i.e. with the terminating NULL there is an even number of bytes) this should be padded out so the overall token size is even.

Constant Shift : 0001 CCCC CCCC CCCC $1000-$1FFF

This allows the extension of constants from 15 bits to 27 bits, e.g. enough to encompass the address range of a 65816 CPU. When there is a constant shift, it is shifted left 15 times and added to the 15 bit standard constant. This clears the constant shift.

Keywords : 001T TTTK KKKK KKKK $2000-$3FFF

These represent tokenised keywords. The numbers are not continuous though the K KKKK KKKK value is – this is the actual keyword token. TTTT identifies it’s type as follows.

0000-0111 Binary Operators. 0000 is the lowest level

(e.g. AND OR NOT)

1000-1100 Unused at present

1101 Unary function. Note some binary functions such

as ! ? and – are also unary ones.

1110 Syntactic only keywords – things that aren’t a

command, such as , ; and TO

1111 An executable keyword command LET LIST etc

Identifier : 01ET ACCC CCCC CCCC $4000-$7FFF

The identifier token has 2 ASCII values packed into a 16 bit word.

E is a continuation bit, if it is zero it marks the last token in the identifier

T is a type bit, if 0 it is an integer, if 1 it is a string

A indicates the presence of an array.

The constant CCC CCCC CCCC is built from two characters, c1 and c2, the value being c1 + 45 \* c2. The values c1 and c2 are as follows : 1-26 are A-Z 27-36 are 0-9 and 0 is used for padding.

Tokens are sequenced together to form longer identifiers, the last one (which may also be the first one) has E = 0. A and T are the same values for every token entry.

Constant : 1CCC CCCC CCCC CCCC $8000-$FFFF

Normally represents the constant 0-32767. The constant shift can extend this to 2^27-1. This shift is always zeroed after being applied.

**Program Storage**

Programs are a collection of records in line number order. The header is 2 words ; firstly an ***offset*** to the next line. If this is zero this indicates the program end. Following that is the tokenised line number, from 0-32767 – this has bit 15 set as per the tokenised constant.

Following that is a sequence of tokens ending in the token $0000.

**Variable Storage**

Variable storage immediately follows the end of program storage.

Each record occupies 6 information bytes and data, depending on what is being stored.

+0..1 The *address* of the next variable in the list

+2..3 The *address* of the identifier as a sequence of tokens (stored in program code normally).

+4..5 The largest index of the array, or 0000 for a non-array.

+6.. Data. This is either an array of 32 bit integers or an array

of 16 bit string addresses. For non arrays there is 1 value,

for arrays there is (1+largest) values.

All addresses are absolute addresses, so if the BASIC workspace area is 3000-C000, then those addresses are independent of that. Variable memory is cleared when

For ease of readability there is a gap of 2 words containing the values $EEEEEEEE between the program code and the start of variable / data space.

Variables are stored in linked lists, ending with a next variable address (the link) of $0000. There are 4 arrays of up to 16 hash entries for each of the four types, the hash entry is obtained by exclusive-oring the upper and lower bytes of the first token.

**Memory Usage**

All are offset from the base address.

+0000 Module identifier “BASC” (in that order)

+0004 Next free byte after program store (reset on

CLEAR, any program editing, NEW etc.)

+0006 Byte after last used byte ; top of memory.

+0008..+001F *Unused*

+0020-003F Up to 16 linked list pointers (Integer)

+0040-005F Up to 16 linked list pointers (String)

+0060-007F Up to 16 linked list pointers (Integer Array)

+0080-009F Up to 16 linked list pointers (String Array)

+00A0-00BF *Unused.*

+00C0 First program record (offset to next)